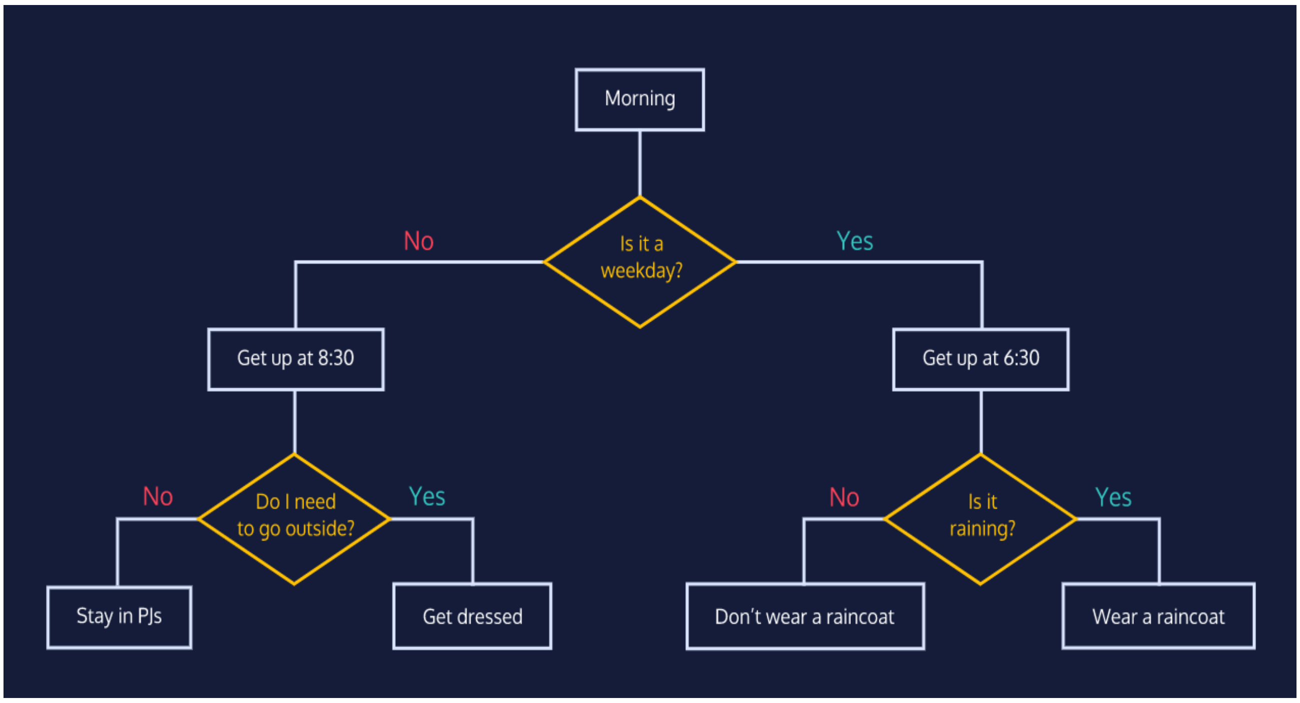
Python
Lecture 2

Control Flow

1.1 Introduction to Control Flow



Imagine waking up in the morning.

You wake up and think, “Ugh. Is it a weekday?”

If so, you have to get up and get dressed and get ready for work or school. If not, you can sleep in a bit longer and catch a couple extra Z’s. But alas, it is a weekday, so you are up and dressed and you go to look outside, “What’s the weather like? Do I need an umbrella?”

These questions and decisions control the flow of your morning, each step and result is a product of the conditions of the day and your surroundings. Your computer, just like you, goes through a similar flow every time it executes code. A program will run (wake up) and start moving through its checklists, is this condition met, is that condition met, okay let’s execute this code and return that value.

This is the *control flow* of your program. In Python, your script will execute from the top down, until there is nothing left to run. It is your job to include gateways, known as conditional statements, to tell the computer when it should execute certain blocks of code. If these conditions are met, then run this function.

Over the course of this lesson, you will learn how to build conditional statements using boolean expressions, and manage the control flow in your code.

1.2 Relational Operators: Equals and Not Equals

Relational operators compare two items and return either `True` or `False`. For this reason, you will sometimes hear them called *comparators*.

The two relational operators we'll cover first are:

- Equals: `==`
- Not equals: `!=`

These operators compare two items and return `True` or `False` if they are equal or not.

We can create boolean expressions by comparing two values using these operators:

```
1 == 1      # True

2 != 4      # True

3 == 5      # False

'7' == 7    # False
```

Each of these is an example of a boolean expression.

Why is the last statement false? The `''` marks in `'7'` make it a string, which is different from the integer value `7`, so they are not equal. When using relational operators it is important to always be mindful of type.

Example

Determine if the following boolean expressions are `True` or `False`. Input your answer as `True` or `False` in the appropriate variable to the right.

Statement one:

```
(5 * 2) - 1 == 8 + 1
```

Statement two:

```
13 - 6 != (3 * 2) + 1
```

Statement three:

```
3 * (2 - 1) == 4 - 1
```

1.3 Boolean Variables

Before we go any further, let's talk a little bit about `True` and `False`. You may notice that when you type them in the code editor (with uppercase T and F), they appear in a different color than variables or strings. This is because `True` and `False` are their own special type: `bool`.

`True` and `False` are the only `bool` types, and any variable that is assigned one of these values is called a *boolean variable*.

Boolean variables can be created in several ways. The easiest way is to simply assign `True` or `False` to a variable:

```
set_to_true = True
set_to_false = False
```

You can also set a variable equal to a boolean expression.

```
bool_one = 5 != 7
bool_two = 1 + 1 != 2
bool_three = 3 * 3 == 9
```

These variables now contain boolean values, so when you reference them they will only return the `True` or `False` values of the expression they were assigned.

```
print(bool_one)    # True

print(bool_two)    # False

print(bool_three)  # True
```

Example

```
bool1 = "true"
print(type(bool1))

bool2 = True
print(type(bool2))
```

1.4 If Statement

Understanding boolean variables and expressions is essential because they are the building blocks of *conditional statements*.

Recall the waking-up example from the beginning of this lesson. The decision-making process of “Is it raining? If so, bring an umbrella” is a conditional statement.

Here it is phrased in a different way:

```
If it is raining, then bring an umbrella.
```

Can you pick out the boolean expression here?

Right, `"it is raining"` is the boolean expression, and this conditional statement is checking to see if it is True.

If `"it is raining" == True` then the rest of the conditional statement will be executed and you will bring an umbrella.

This is the form of a conditional statement:

```
If [it is raining], then [bring an umbrella]
```

In Python, it looks very similar:

```
if is_raining:
  print("bring an umbrella")
```

You’ll notice that instead of “then” we have a colon, `:`. That tells the computer that what’s coming next is what should be executed if the condition is met.

Let’s take a look at another conditional statement:

```
if 2 == 4 - 2:
  print("apple")
```

Example

```
# Enter a user name here, make sure to make it a string
user_name = "jo_bond007"

if user_name == "jo_bond007":
  print("Welcome to the secret hide out Bond!")
if user_name == "evil_doc":
  print("INTRUDER ALERT!!! The evil doc is here!")
```

1.5 Relational Operators II

Now that we've added conditional statements to our toolkit for building control flow, let's explore more ways to create boolean expressions. So far we know two relational operators, equals and not equals, but there are a ton (well, four) more:

- `>` greater than
- `>=` greater than or equal to
- `<` less than
- `<=` less than or equal to

Let's say we're running a movie streaming platform and we want to write a program that checks if our users are over 13 when showing them a PG-13 movie. We could write something like:

```
if age <= 13:
  print("Sorry, parental control required")
```

This function will take the user's age and compare it to the number 13. If `age` is less than or equal to 13, it will print out a message.

Let's try some more!

Example

```
x = 20
y = 20

# Write the first if statement here:
if x == y:
  print("These numbers are the same")


credits = 120

# Write the second if statement here:
if credits >= 120:
  print("You have enough credits to graduate!")
```

1.6 Boolean Operators: and

Often, the conditions you want to check in your conditional statement will require more than one boolean expression to cover. In these cases, you can build larger boolean expressions using *boolean operators*. These operators (also known as *logical operators*) combine smaller boolean expressions into larger boolean expressions.

There are three boolean operators that we will cover:

- `and`
- `or`
- `not`

Let's start with `and`.

`and` combines two boolean expressions and evaluates as `True` if both its components are `True`, but `False` otherwise.

Consider the example:

```
Oranges are a fruit and carrots are a vegetable.
```

This boolean expression is comprised of two smaller expressions, `oranges are a fruit` and `carrots are a vegetable`, both of which are `True` and connected by the boolean operator `and`, so the entire expression is `True`.

Let's look at an example of some AND statements in Python:

```
(1 + 1 == 2) and (2 + 2 == 4)   # True

(1 > 9) and (5 != 6)            # False

(1 + 1 == 2) and (2 < 1)        # False

(0 == 10) and (1 + 1 == 1)      # False
```

Notice that in the second and third examples, even though part of the expression is `True`, the entire expression as a whole is `False` because the other statement is False. The fourth statement is also `False` because both components are `False`.

Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
(2 + 2 + 2 >= 6) and (-1 * -1 < 0)
```

Statement two:

```
(4 * 2 <= 8) and (7 - 1 == 6)
```

Example

```
credits = 120
gpa = 3.4

if credits >= 120 and gpa >= 2.0:
  print("You meet the requirements to graduate!")
```

1.7 Boolean Operators: or

The boolean operator `or` combines two expressions into a larger expression that is `True` if either component is `True`.

Consider the statement

```
Oranges are a fruit or apples are a vegetable.
```

This statement is composed of two expressions: `oranges are a fruit` which is `True` and `apples are a vegetable` which is `False`. Because the two expressions are connected by the `or` operator, the entire statement is `True`. Only one component needs to be `True` for an `or` statement to be `True`.

In English, `or` implies that if one component is `True`, then the other component must be `False`. This is not true in Python. If an `or` statement has two `True` components, it is also `True`.

Let's take a look at a couple of examples in Python:

```
True or (3 + 4 == 7)    # True
(1 - 1 == 0) or False   # True
(2 < 0) or True         # True
(3 == 8) or (3 > 4)     # False
```

Notice that each `or` statement that has at least one `True` component is `True`, but the final statement has two `False` components, so it is `False`.

Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
(2 - 1 > 3) or (-5 * 2 == -10)
```

Statement two:

```
(9 + 5 <= 15) or (7 != 4 + 3)
```

Example

```
credits = 118
gpa = 2.0

if credits >= 120 or gpa >= 2.0:
  print("You have met at least one of the requirements.")
```

1.8 Boolean Operators: not

The final boolean operator we will cover is `not`. This operator is straightforward: when applied to any boolean expression it reverses the boolean value. So if we have a `True` statement and apply a `not` operator we get a `False` statement.

```
not True == False
not False == True
```

Consider the following statement:

```
Oranges are not a fruit.
```

Here, we took the `True` statement `oranges are a fruit` and added a `not` operator to make the `False` statement `oranges are not a fruit`.

This example in English is slightly different from the way it would appear in Python because in Python we add the `not` operator to the very beginning of the statement. Let’s take a look at some of those:

```
not 1 + 1 == 2  # False
not 7 < 0       # True
```

Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
not (4 + 5 <= 9)
```

Statement two:

```
not (8 * 2) != 20 - 4
```

Example

```
if not credits >= 120:
  print("You do not have enough credits to graduate.")

if not gpa >= 2.0:
  print("Your GPA is not high enough to graduate.")

if not credits >= 120 and not gpa >= 2.0:
  print("You do not meet either requirement to graduate!")
```

1.9 Else Statements

As you can tell from your work with *Calvin Coolidge's Cool College*, once you start including lots of `if` statements in a function the code becomes a little cluttered and clunky. Luckily, there are other tools we can use to build control flow.

`else` statements allow us to elegantly describe what we want our code to do when certain conditions are **not** met.

`else` statements always appear in conjunction with `if` statements. Consider our waking-up example to see how this works:

```
if weekday:
  print("wake up at 6:30")
else:
  print("sleep in")
```

In this way, we can build if statements that execute different code if conditions are or are not met. This prevents us from needing to write `if` statements for each possible condition, we can instead write a blanket `else` statement for all the times the condition is not met.

Let's return to our `if` statement for our movie streaming platform. Previously, all it did was check if the user's age was over `13` and if so, print out a message. We can use an `else` statement to return a message in the event the user is too young to watch the movie.

```
if age >= 13:
  print("Access granted.")
else:
  print("Sorry, you must be 13 or older to watch this movie.")
```

Example

```
credits = 120
gpa = 1.9

if (credits >= 120) and (gpa >= 2.0):
  print("You meet the requirements to graduate!")
else:
  print("You do not meet the requirements to graduate.")
```

1.10 Else If Statements

We have `if` statements, we have `else` statements, we can also have `elif` statements.

Now you may be asking yourself, what the heck is an `elif` statement? It's exactly what it sounds like, "else if". An `elif` statement checks another condition after the previous `if` statements conditions aren't met.

We can use `elif` statements to control the order we want our program to check each of our conditional statements. First, the `if` statement is checked, then each `elif` statement is checked from top to bottom, then finally the `else` code is executed if none of the previous conditions have been met.

Let's take a look at this in practice. The following `if` statement will display a "thank you" message after someone donates to a charity; there will be a curated message based on how much was donated.

```
print("Thank you for the donation!")

if donation >= 1000:
  print("You've achieved platinum status")
elif donation >= 500:
  print("You've achieved gold donor status")
elif donation >= 100:
  print("You've achieved silver donor status")
else:
  print("You've achieved bronze donor status")
```

Take a second to think about this function. What would happen if all of the `elif` statements were simply `if` statements? If you donated $1100.00, then the first three messages would all print because each `if` condition had been met.

But because we used `elif` statements, it checks each condition sequentially and only prints one message. If I donate $600.00, the code first checks if that is over 1000, which it is not, then it checks if it's over 500, which it is, so it prints that message, then because all of the other statements are `elif` and `else`, none of them get checked and no more messages get printed.

Try your hand at some other `elif` statements.

Example

AfriHub has noticed that students prefer to get letter grades.

Write an `if`/`elif`/`else` statement that:

- If `grade` is 90 or higher, print `"A"`
- Else if `grade` is 80 or higher, print `"B"`
- Else if `grade` is 70 or higher, print `"C"`
- Else if `grade` is 60 or higher, print `"D"`
- Else, print `"F"`

```python
grade = 86
if grade >= 90:
  print("A")
elif grade >= 80:
  print("B")
elif grade >= 70:
  print("C")
elif grade >= 60:
  print("D")
else:
  print("F")
```

Example 2

Little Codey is an interplanetary space boxer, who is trying to win championship belts for various weight categories on other planets within the solar system.

Write a **space.py** program that helps him keep track of his target weight by:

1. Checks which number `planet` is equal to.

2. It should then compute his weight on the destination planet.

Here is the table of conversion:

#	Planet	Relative Gravity
1	Venus	0.91
2	Mars	0.38
3	Jupiter	2.34
4	Saturn	1.06
5	Uranus	0.92
6	Neptune	1.19

Complete the code

```
print("I have information for the following planets:\n")

print("   1. Venus   2. Mars    3. Jupiter")
print("   4. Saturn  5. Uranus  6. Neptune\n")

weight = 185
planet = 3

# Write an if statement below:
```

Lecture 2 Project

Project 1

Magic 8-Ball

The Magic 8-Ball is a popular toy developed in the 1950s for fortune-telling or advice seeking.

Write a **magic8.py** Python program that can answer any “Yes” or “No” question with a different fortune each time it executes.

We’ll be using the following 9 possible answers for our Magic 8-Ball:

- `Yes - definitely.`
- `It is decidedly so.`
- `Without a doubt.`
- `Reply hazy, try again.`
- `Ask again later.`
- `Better not tell you now.`
- `My sources say no.`
- `Outlook not so good.`
- `Very doubtful.`

The output of the program will have the following format:

```
[Name] asks: [Question]
Magic 8-Ball’s answer: [Answer]
```

For example:

```
Joe asks: Is this real life?
Magic 8-Ball's answer: Better not tell you now
```

Take these into consideration:

1. If the asker does not provide a name, such that the value of `name` is an empty string? If the `name` string is empty, the output of the program looks like the following:

```
 asks: Will I win the lottery?
Magic 8 Ball's answer: Outlook not so good
```

As you can see, the formatting of the output can use some improvement when there is no name provided.

We can address this by printing out just the question, such that it looks like the following:

```
Question: Will I win the lottery?
Magic 8-Ball's answer: Outlook not so good
```

You can implement this by creating an `if`/`else` statement such that:

- If the name is an empty string, it will only print the question.
- Else, the player’s name and question are both printed.

2. What if the `question` string is empty? If the user does not provide any question, then the Magic 8-Ball cannot provide a fortune, otherwise, the fabric of reality is at risk!

   To ensure that the fabric of reality is safe, we can create an `if`/`else` statement where:

   - If the question is an empty string, print out a message to the user.
   - Else, print the name and question, with the Magic 8-Ball’s answer.

Project 2

Sal's Shipping

Sal runs the biggest shipping company in the tri-county area, Sal's Shippers. Sal wants to make sure that every single one of his customers has the best, and most affordable experience shipping their packages.

In this project, you'll build a program that will take the weight of a package and determine the cheapest way to ship that package using Sal's Shippers.

Sal's Shippers has several different options for a customer to ship their package:

- Ground Shipping, which is a small flat charge plus a rate based on the weight of your package.
- Ground Shipping Premium, which is a much higher flat charge, but you aren't charged for weight.
- Drone Shipping (new), which has no flat charge, but the rate based on weight is triple the rate of ground shipping.

Here are the prices:

Ground Shipping

Weight of Package	Price per Pound	Flat Charge
2 lb or less	$1.50	$20.00
Over 2 lb but less than or equal to 6 lb	$3.00	$20.00
Over 6 lb but less than or equal to 10 lb	$4.00	$20.00
Over 10 lb	$4.75	$20.00

Ground Shipping Premium

Flat charge: $125.00

Drone Shipping

Weight of Package	Price per Pound	Flat Charge
2 lb or less	$4.50	$0.00
Over 2 lb but less than or equal to 6 lb	$9.00	$0.00
Over 6 lb but less than or equal to 10 lb	$12.00	$0.00
Over 10 lb	$14.25	$0.00

Write a **shipping.py** Python program that asks the user for the weight of their package and then tells them which method of shipping is cheapest and how much it will cost to ship their package using Sal's Shippers.

Follow these steps and complete the project

Ground Shipping:

1) First things first, define a `weight` variable and set it equal to any number.

2) Next, we need to know how much it costs to ship a package of given weight by normal ground shipping based on the "Ground shipping" table above.

   Write a comment that says "Ground Shipping".

   Create an `if`/`elif`/`else` statement for the cost of ground shipping. It should check for `weight`, and print the `cost` of shipping a package of that weight.

3) A package that weighs 8.4 pounds should cost \$53.60 to ship with normal ground shipping:

   $$8.4\ lb \times \$4.00 + \$20.00 = \$53.60$$

   Test that your ground shipping function gets the same value.

Ground Shipping Premium:

4) We'll also need to make sure we include the price of premium ground shipping in our code.

   Create a variable for the cost of premium ground shipping.

   Note: This does not need to be an `if` statement because the price of premium ground shipping does not change with the weight of the package

5) Print it out for the user just in case they forgot!

Drone Shipping:

6) Write a comment for this section of the code, “Drone Shipping”.

   Create an `if`/`elif`/`else` statement for the cost of drone shipping. This statement should check against `weight` and print the `cost` of shipping a package of that weight.

7) A package that weighs 1.5 pounds should cost $6.75 to ship by drone:

   $$1.5\ lb \times \$4.50 + \$0.00 = \$6.75$$

   Test that your drone shipping function gets the same value.

Solution:

8) Great job! Now, test everything one more time!

   What is the cheapest method of shipping a 4.8 pound package and how much would it cost?

   What is the cheapest method of shipping a 41.5 pound package and how much would it cost?